

# COMP 322: Fundamentals of Parallel Programming

## Lecture 31: TF-IDF and Page Rank Algorithms using Map-Reduce Parallelism

Mack Joyner and Zoran Budimlić  
{mjoyner, zoran}@rice.edu

<http://comp322.rice.edu>



# Worksheet #30 solution: Variant of Word Count

---

```
1. JavaRDD<String> file = context.textFile(inputFile);
2. // Change w.r.t. slide 13: replace s by s.length()
3. JavaPairRDD<???, Integer> counter =
4.     file.flatMap(s -> Arrays.asList(s.split(" ")))
5.         .mapToPair(s -> new Tuple2<>(s.length(), 1))
6.         .reduceByKey((a, b) -> a + b);

8. counter.collect().foreach(System.out::println);
```

a) In the space below, indicate what type should be provided instead of ??? in line 3.

**Integer**

b) Also, explain what this program computes.

**The frequencies of word lengths**



# Background for TF-IDF algorithm (can be implemented as M-R jobs in Hadoop or Spark)

---

- Goal: Given a document,  $D_0$ , find most similar documents in a collection of documents,  $D_1, \dots, D_N$
- Approach: model each document as a multiset of terms (“bag of words”) and use word frequencies to guide similarity search. Let  $TERM_1, TERM_2, \dots$  represent all the terms across all documents
- Definitions
  - TF(i,j) = total frequency (count) of  $TERM_i$  in document  $D_j$ 
    - Measure of significant terms in a document
  - DF(i) = number of documents that contain  $TERM_i$
  - IDF(i) =  $N / DF(i)$ 
    - Measure of how common or rare a term is across all documents
  - Commonly used weight of  $TERM_i$  in  $D_j = TF(i,j) * \log (IDF(i))$
- See <https://en.wikipedia.org/wiki/Tf-idf> for more background



# Map-Reduce Job 1: Computing TF

---

- **Map task**
  - Input:  $(D_i, TERM_j)$  pairs for all terms in documents (including duplicates)
  - Output:  $((D_i, TERM_j), 1)$  for each input pair
- **Reduce task**
  - Use SUM as reduce operator
  - Outputs  $((D_i, TERM_j), TF(i,j))$



# Map-Reduce Job 2: Computing DF

---

- **Map task**
  - Input:  $(D_i, \text{TERM}_j)$  pairs for all terms in documents (without duplicates)
  - Output: for each document,  $(\text{TERM}_j, 1)$  for occurrence of  $\text{TERM}_j$
- **Reduce task**
  - Use **SUM** as reduce operator
  - Outputs  $(\text{TERM}_j, \text{DF}(j))$
- **IDF can be easily computed from DF using a map task**



# Background for PageRank algorithm and its implementation in Spark

- Give pages ranks (scores) based on links to them
  - Links from many pages → high rank
  - Link from a high-rank page → high rank
- Needs an iterative map-reduce algorithm
- Good match for Spark's in-memory processing capabilities



Acknowledgment: slides for this topic were taken from “Parallel Programming With Spark” lecture by Prof. Matei Zaharia, Stanford University



# Algorithm

Start each page at a rank of 1

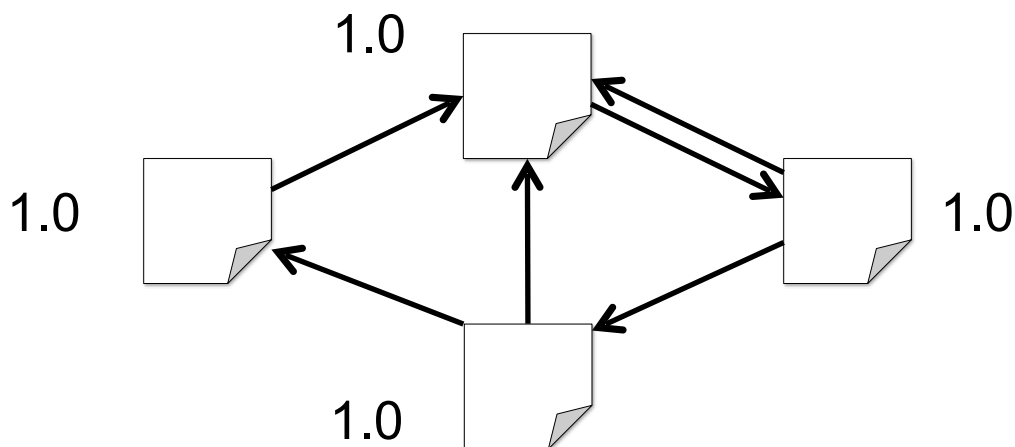
FOR (iter = ...) {

1. On each iteration, have each page *A contribute* to the rank of B when there is a link (edge) from A to B

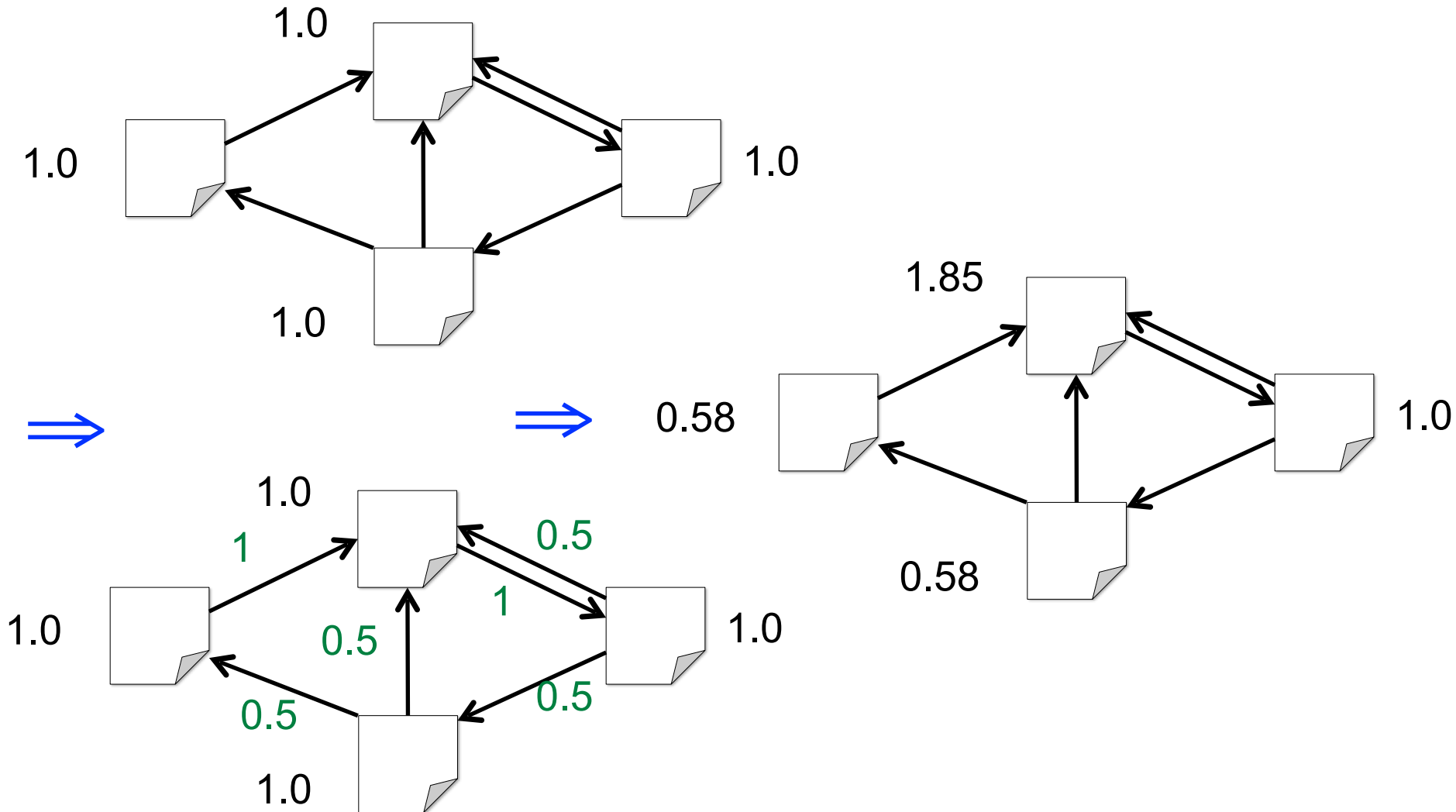
$$\text{CONTRIBS}(B) += \text{RANK}(A) / \text{DEST\_COUNT}(A)$$

2. Update all page ranks to  $\text{RANK}(B) = 0.15 + 0.85 \times \text{CONTRIBS}(B)$

}

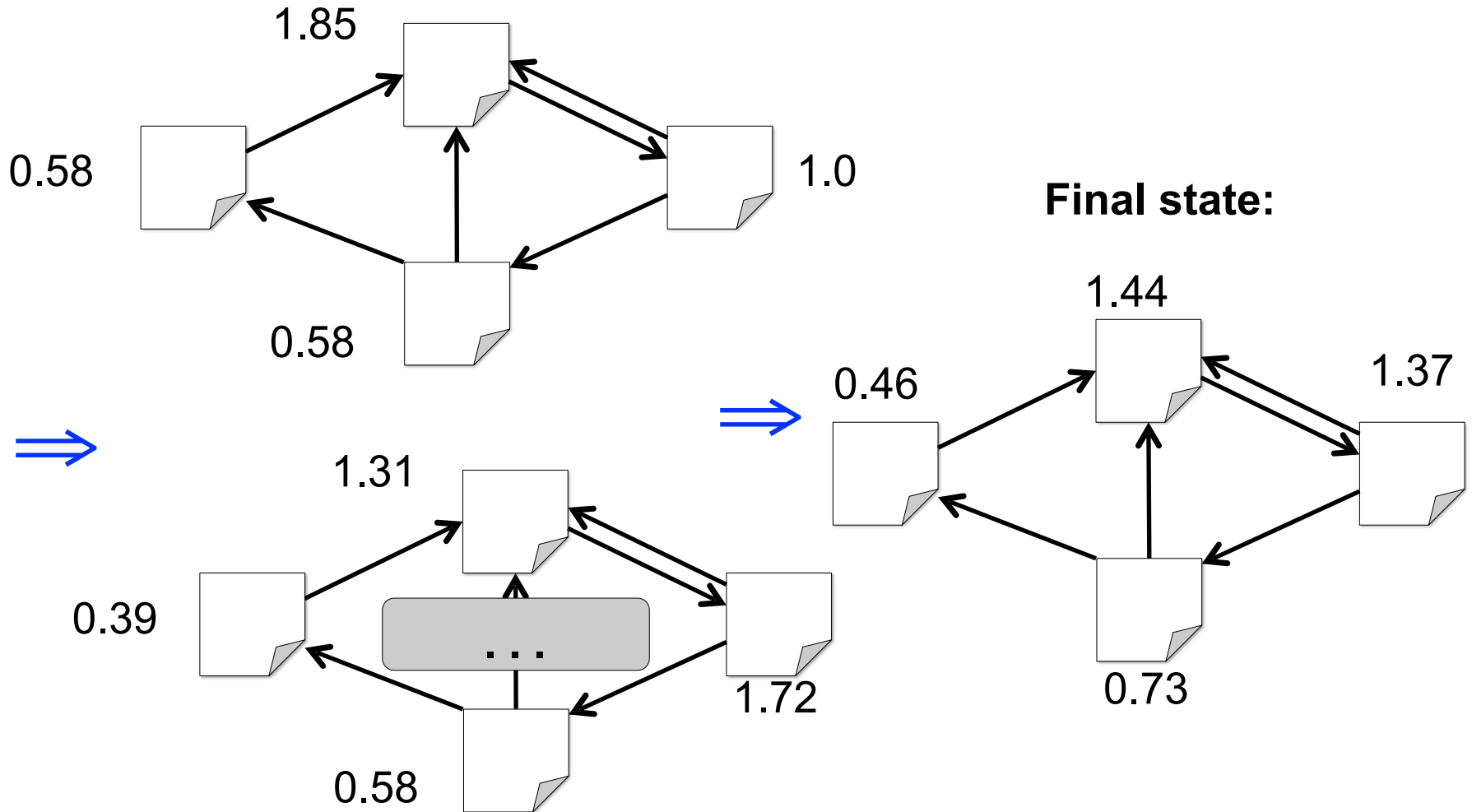


# Example: First Iteration





# Example: Successive Iterations



# Announcements and Reminders

---

- **Checkpoint 1 for Homework 4 is due this Wednesday by 11:59pm (April 3rd)**
- **Quiz for Unit 8 is due by Friday, April 5th at 11:59pm**
- **Final exam (Exam 2) is scheduled at 9am - 12noon on Wednesday, May 1st (scope of exam is limited to lectures 18 - 38)**

